BASIC/S 3.1 COMPILER
BY: BILL STOCKWELL
FOR MOD I AND MOD III

Q.S.D.

BASIC/S

(C) 1981

BASIC/S©

A BASIC COMPILER
FOR MOD I & III

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    BASIC/S COMPILER                          *
*              (C) 1981 by Bill Stockwell and QSD              *
*                 -Version 3.1 for Mod I and III-             *
*                    -All Rights Reserved-                     *
*  Published by: Quality Software Distributors / Dallas,Tx    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

INTRODUCTION :

BASIC/S   is a BASIC program which compiles a subset of TRS-80(c)
Disk Basic into Z80 machine code. The machine code thus created may be
saved on disk as a command file. The program is easy to use, and may be
run under almost any Mod I/Mod III DOS.

GETTING STARTED :

-- MOD I --

Boot up the disk you recieved with this package. It will then prompt
you for a destination drive. This can be drive 0,1,2, or 3. Insert a
SYSTEM disk of your choice (LDOS, TRSDOS, Newdos, Newdos/80, DOSPLUS, or
VTOS). You may use a formatted disk if you wish also, as long as you
have a more than one drive, (one to put the SYSTEM in). Answer the
prompts you see on the monitor, and in just a few moments you will have
BASIC/S installed on the SYSTEM of your choice. It is VERY easy to do.
Your destination disk MUST be formatted or be a SYSTEM disk!! The only
files that you actually need to copy over to use the compiler are
BASIC/S and COMPILER/DAT. The other files are either utilities or
example files for you to compile. The reason we mention this is that not
all of the files will fit on one 35/40 track single density disk, if all
you have is one drive. Our loader program (by Kim Watt), is intelligent
enough, however, to sense that you do not have enough room! It then will
allow you to selectivly place the files where you want them.

If you are using a ONE DRIVE SYSTEM (single density 35/40 track), we
suggest that you prepare two SYSTEM disks that have been "stripped"
down. Transfer BASIC/S to one of them, and everything else to the other
disk. When you want to compile one of your own programs, place it on a
system disk along with COMPILER/DAT. Load your first disk that has
BASIC/S on it. Once BASIC/S is in memory and running, you may switch
diskettes. Remember.... COMPILER/DAT MUST BE ON-LINE AT ALL TIMES WHILE
COMPILING!

BASIC/S   SELECT/BAS
COMPILER/DAT   LOOK/BAS
SHELL/BAS   SPACEWAR/BAS
TEST/DAT   COMPARE/BAS

-- MOD III --

This disk is in Model I single density format. Simply place the disk you
have recieved in DRIVE 1, and use CONVERT, to "convert" the disk to Mod
III format (TRSDOS 1.1 - 1.3) If you are using DOSPLUS, follow their
respective directions for "converting" over. With LDOS 5.1/III, use the
REPAIR command, and this disk will be directly readable by LDOS, even
though it is in Single Density. You may then COPY them over. REPAIR :1
(ALIEN).   Easy to do! There is plenty of room on a Mod III 40 track
double density diskette for the whole set of programs, although
remember.... all you REALLY need on the BASIC/S disk that you make for
day to day use is BASIC/S and COMPILER/DAT.

Getting Started Using BASIC/S:

IMPORTANT!! There is a variable in BASIC/S, in the very first line,
which tells BASIC/S what disk operating system you are using. Currently,
this is used so that LOF calculations will be done properly - ie when
you compile a program that does an LOF calculation, it is important for
the compiler to know what DOS is being used so that this calculation
will be done properly. (The assumption is that you will run your /CMD
files under the same system that they were compiled on. If this is not
true, you need to change the variable as explained below and recompile
under the other DOS). The variable in question is KS, and is found at
the end of line 15 of BASIC/S. It is now set as KS = 0. This is the
correct setting for all DOS's EXCEPT for DOSPLUS(c) and LDOS(c).   KS
should be set to four (4) if you are using DOSPLUS, and to five (5) for
LDOS. Be certain to make this change in your copy of BASIC/S before
continuing, ONLY IF YOU ARE USING ONE OF THESE SYSTEMS!!

Edit Line 15, end of line: (now says, :KS=0)

        Dosplus - :KS=4   LDOS - :KS=5

On the disk you receive, there will be just one copy of BASIC/S, one
of COMPILER/DAT, and some supplementary demo and utility files. Copy
these to a disk of your own.

It   is   a   good   idea at this time to compile one of the sample
programs on the disk. SHELL/BAS, SELECT/BAS, LOOK/BAS, SPACEWAR/BAS,
and COMPARE/BAS are all   BASIC/S compilable. SHELL/BAS is a Shell
Metzner sort program which will sort an ASCII sequential disk file of
up to 79 strings; after you compile it, you invoke it via SHELL
OUTPUT=INPUT from DOS READY mode, where INPUT is the file you want to
sort, and OUTPUT is a new file which you want the sorted file to be
written to. Don't try to run SHELL/BAS from BASIC (as is). It checks
the DOS command buffer at 4318H for the file specifications, which
will not be meaningful from BASIC. TEST/DAT is a file for SHELL to
sort.   After compiling SHELL/BAS execute the /CMD file via SHELL
OUT=TEST/DAT (where OUT is your output file). To compile SHELL/BAS, you
should get into Basic and RUN"BASIC/S", making sure that COMPILER/DAT
and SHELL/BAS are on line ; when BASIC/S asks "Files, options ?",
respond with:

          SHELL/BAS,SHELL/CMD,,56000 <enter>

This way, your command file will be placed into high memory, making room
for a string array of dimension 79 (T$) in low memory. If you do not
specify a starting address (which you normally wouldn't), it will
default to 5200H, which means that the T$ array will be placed in high
memory, where there is room for only 37 strings (and that's counting all
the way up to FFFFH !). No problem if your file is no more than 37
strings long AND you have no high memory drivers in place; otherwise...
See below for more information on running BASIC/S.

SELECT is a program for making small (selective) zaps to COMPILER/DAT
(the module which the compiler uses to get most of the object code
which is used to create your /CMD files). Practice using SELECT only
on a COPY of COMPILER/DAT!! You will be told when, if, and how to use
it if it ever becomes necessary. LOOK allows you to "look" at
selected records in COMPILER/DAT without changing them, in order to
verify a zap for example. COMPARE/BAS allows you to compare 2 files to
see if they are the same.

SPACEWAR/BAS is a fast paced, real time shoot the Klingons game. You can
run it in Basic or as a /CMD file after you compile it, but it runs MUCH
faster compiled!

For the most part, these five programs allow you to become familiar
with the rather restrictive syntax of BASIC/S.

The version of BASIC which is supported is a subset of Disk Basic.
Only simple expressions and variable names are allowed, but most of
the features and built-in functions of Level II are implemented, along
with the essential elements of sequential and random disk I/O.

Note: Unlike regular BASIC, programs compiled by BASIC/S do NOT have
any initialization of variables done. Thus numeric variables do not
start out as zero, or strings as null. (See the CLEAR statement,
however). One advantage of this approach is that one compiled program
can invoke another (using the RUN statement) and all variables will be
preserved.

Use of constants in BASIC/S is somewhat restricted; many statements
allow (real or integer) constants ; most statements do NOT allow string
constants. See the section below on the individual statements for more
details.

You may have multiple statements per line; the only restriction
here is that IF, GOTO, and GOSUB statements must begin the line they
are on. Spacing is critical when writing a program to be compiled by
BASIC/S; in general, use spaces only to separate keywords from
identifiers (FOR N%=A% TO B% rather than FOR N%=A%TOB%).

Look over some of the sample programs on the disk to see how
statements are to be coded. The syntax must be followed....

                -------> E X A C T L Y !! <-------

The compiler allows the following variable names (all single
letters): integers A% thru Z%, reals A-Z, and strings A$ thru Z$.
Also, you may dimension arrays of any of these three types, and your
array names can be any length, with every character significant.

See the DIM statement for more on this.

BASIC/S does NOT allow free conversion between real and integer
variables - you MUST use the built-in functions CINT and CSNG in order
to convert between them. See below.

REQUIREMENTS :

A TRS-80(c) Mod I or III with at least one drive and 48K.
Repeat... 48K.

RUNNING THE PROGRAM :

Load BASIC (whatever DOS you are using), no need to set memory size, and
RUN"BASIC/S". Be sure you have set KS if you are using LDOS or Dosplus.
Be sure COMPILER/DAT is on line, as well as the program you wish to
compile (saved in ASCII!).
REPEAT.... your program to compile MUST be saved in ASCII!

e.g.   SAVE"FILENAME/BAS",A <enter>

                (the /BAS extension is NOT required)

Now BASIC/S will ask:

                Files, options ?

The typical response will be in the form:

                SOURCE,OBJECT

e.g. TEST,TEST/CMD where TEST is the name of the ascii Basic program you
want to compile, and TEST/CMD is the name of the load module you want to
create. You may specify drive specs after either file name. It is best
if the OBJECT file does not already exist. If it does exist, BASIC/S
will kill it before continuing. No big deal, but it takes a little
longer.

Two other parameters may be specified here. The first will produce
output to the system line printer (in the form of source code and
errors), while the second will tell BASIC/S where the load module's
start point is to be. To specify line printer output, just put a * (or
*PR, or *pr, or *anything) as the third parameter. The address, if
present, should be a decimal integer in the fourth position. It may be
positive or negative - Basic/s will respond correctly either way. Thus,
complete syntax to the "Files ?" question is:

                SOURCEFILE,OBJECTFILE,<*PR>,<ADDRESS>

where the brackets "< >" are NOT to be typed, but indicate optional
entries. If no printer output is wanted, but an address is to be
specified, then the third parameter should be null - ie, present, but

null or blank as indicated by two adjacent commas.

                TEST/BAS,TEST/CMD:1,,<ADDRESS>

The compiler gets much of the data needed to compile your program from
a random access disk file (COMPILER/DAT). Be sure this file is on line
when you use BASIC/S.

Note:
When running BASIC/S compiled programs on a Mod III using Mod III
TRSDOS,
you must execute the /CMD files BASIC/S creates from BASIC -
enter the program from BASIC in the following way:
        (from BASIC)   CMD"I","PROGNAME/CMD" <enter>
Actually, this is not always necessary, but for some unknown
reason, BASIC/S /CMD files sometimes crash when exceuted from
DOS READY under Mod III TRSDOS, whereas they do just fine when
run from BASIC like this. NO other DOS, Mod I or III, has this
restriction.

THE BASIC/S SUBSET (Statements supported under BASIC/S) :

PRINT
-----
        followed by a SINGLE variable name, or an expression in
        quotes. Thus :

                PRINT A% or
                PRINT"Message"

        Also, you may use a semi-colon after anything being
        printed in order to suppress the carriage return.

            PRINT@ is also supported -- just set any integer
        variable to the value of the location to be printed
        at, and you may then use any of the above forms with
        it. Thus :

                PRINT@N%,"TRS-80";

LPRINT

        Syntax for LPRINT is in every way the same as for
        PRINT, except of course that LPRINT@ has no meaning.

INPUT

        You may input a single variable, of any type. You may
        not input a list of variables, but INPUT"PROMPT";A is
        supported (or A%, or A$). Note: Spacing is important
        in BASIC/S. Do not run keywords and variable names

together -- use a single space in between them.
When executing a Basic/s compiled program, if input
is requested, hitting the <break> key will cause an
exit to DOS READY.
        IMPORTANT: When inputting floating point
variables, you MUST use a decimal point (even if the
number is a whole number). Also, you may not use 'E'
notation on input. If in answer to an input prompt, you
hit <Enter> only, then the variable being inputted
remains unchanged and the program continues (just like
regular BASIC -- and this holds regardless of variable
type (integer, real or string)).

LINE INPUT

        LINE INPUT from the keyboard is supported. Syntax is exactly
        as it is in BASIC. You can even make LINEINPUT one word
        if you like. You may LINE INPUT a real or an interger variable
        if you wish, although this would not work in BASIC.

        e.g.   LINE INPUT A$ or
               LINE INPUT "Prompt";A$ (just like in BASIC)

RUN A$

        This statement allows you to set a string (A$ in this
        case) to any DOS command, or the name of a command file
        you wish to invoke, and to exit the current program and
        have that command executed. Do NOT say RUN"PGM"; this
        will be not be correctly compiled! Also, RUN by itself
        is incorrect.

CLEAR

        This statement, with or without an argument, will cause
        BASIC/S variables to be zeroed out. It depends on where
        your /CMD file starts; if your /CMD file is in low
        memory, then all memory from 41216 (decimal) up to
        HIGH$ will be zeroed out, while otherwise 5200H up to
        D6D8H is zeroed out. This makes sure that your /CMD
        file itself will never be affected, but that your
        variables will be zeroed. This works equally well on
        the Mod I or the Mod III - Basic/s knows which machine
        you are running it on, and will use the correct HIGH$
        for your machine. DATA will also be cleared, and an
        automatic RESTORE done so that the DATA pointer will be
        correct.

GOTO ln

        The GOTO statement. Do not space between the GO and the
        TO. DO space between the GOTO and the line number.

GOSUB ln

        The standard GOSUB statement. Be sure your GOSUB's and
RETURNS match up properly, or your /CMD file may crash.

DEF FN

        This statement works almost exactly as in BASIC, the
only limitations being that the right hand side must be
already handleable by BASIC/S as in a normal assignment
statement, and also only one argument is allowed. Thus
it would be most useful in the case of the target
variable being real, with the right hand side a real
expression (see the section on assignment statements).
But the argument and the target may be any type (real,
integer, or string). Although only one argument is
allowed, you may use any other variables you like on
the right hand side -- but they won't be dummy.
Note : Constants may be used (real and integer,
anyway).

READ / DATA / RESTORE

        Your program may have DATA   statements, containing
integer constants only (as in DATA 1,2,3) -- in all of
your DATA statements you can have a total of 383
integers (no more). It is important that these DATA
statements come before the READ statement(s) that are
to access them (physically before, that is) -- the
compiler generates code to place the data in memory
when the DATA statements are encountered. Syntax for
the READ statement is READ N% -- you can read only a
single integer variable, which would normally be done
in a FOR/TO loop. One big use for this is to poke DATA
for a USR routine into memory. Before BASIC/S allowed
READ/DATA, this process was rather clumsy.

RESTORE
----
        works just like in standard BASIC.

IF
--
        A very restricted IF statement -- you may only compare a
floating point expression with zero, or two strings,
or two simple integers (variables or constants). For
floating points, syntax is:

          IF X<0 THEN 100
      or   IF Z=0 THEN 80
      or   IF SIN(A*B-C)<0 THEN 200
        (more on real expressions later).

The variable must be on the left. For strings, you can
say

            IF A$<B$ THEN 20
     or   IF A$=B$ THEN 100

The compare must be in the '<' direction only, or with
'='. You may check whether a string is null via

          IF A$="" THEN 200 (for example)
but this is the only time you may test a string against
a   constant.

      For integers :
               IF A%=B% THEN 100
        or   IF A%<B% THEN 50
      (and either A% or B% may be an integer constant, as
      in IF A%<72 THEN 200).

        *** Note: GOTO, GOSUB, and IF statements MUST
begin the line that they are on. Also, you may follow
an IF statement with more statements on the same line,
but they will be treated as if they were on succeeding
lines. Thus

          IF X=0 THEN 20:Y=SQR(Z)

would have the effect of IF X=0 THEN 20 ELSE Y=SQR(Z)
(but BASIC/S does not support ELSE). For compatibility
with standard Microsoft Basic, it is best not to follow
IF statements with anything.

FOR/NEXT

        The For/Next loop is implemented for INTEGERS only. You
may code

          FOR A%=B% TO C% (spacing important!)
            ...
          NEXT A%

Constants may be used where B% and C% are indicated,
as long as they are integers (positive, negative, or
zero). Just be sure to use a single space after FOR
and before and after TO. The variable in the NEXT
statement is NOT optional. There is no STEP clause.
FOR/NEXT loops may be (statically) nested.

USR

        A single USR call is allowed. It must be set up by
DEFUSR, and the calling address must be a simple
decimal integer constant. Thus :

          DEFUSR=-1000

        Note: There is no VARPTR statement. However, the

addresses of all simple variables in BASIC/S are   always
the same and may be calculated as follows :

         REALS : If the ascii code for the variable is
                  A, then the   VARPTR will be
                  -11535+4*(A-65).
     INTEGERS : -11406+2*(A-65).
     STRINGS : -23192+256*(A-65).

Strings are stored a little differently than in Level
II. Each string is allocated 256 bytes, the first of
which contains the length of the string (0 to 255) and
the rest of which contain the string itself.   The
Varptr points to the length byte.

Y%=USR(X%)

     This causes the routine whose address was defined by a
     previous DEFUSR statement to be called.   The current
     value of X% is loaded into the HL register pair before
     the call is made, and on return, Y% is given the value
     in the HL register pair. Do not call the ROM routine
     at 0A9A for this. Any integer variables may be used,
     not just X% and Y%. Also, a (decimal) integer
     constant may be used as the argument to be passed.

SET, RESET, and POINT

     Use integers (either variables (followed by %) or
     constants) as the arguments. As with most BASIC/S
     functions, they may not be used in more complex
     expressions. Thus
             SET(X%,20)
             A%=POINT(B%,C%)
     The latter is the only way to access POINT - it cannot
     be invoked in an IF statement.

PEEK and POKE

     Exactly as in Level II, except that
     the arguments   must be integers -- (constants
     or variables). Thus

         A%=PEEK(M%)
         POKE A%,B%
         POKE 15360,191
         Z%=PEEK(14312)

INP and OUT

     Syntax here is just like that for PEEK and POKE,
     i.e. you may use integer variables or constants
     as the arguments (no expressions).

         A%=INP(P%) (input a byte from port P% and
                         store in A%)
         OUT P%,V% (output value V% to port P%)
         OUT 255,1
         S%=INP(232)

AND/OR

     You may use these two functions in order to calculate
     an AND/OR result (for integer variables or constants)
     and store the answer in an integer variable. Thus

             X%=Y% AND 20
             U%=A% OR B%

CLS -- Clear the screen

RND

     Random numbers between 0 and 1 may be generated by the
     statement X=RND(0). The left hand side may be any real
     variable. The argument is not actually required; you
     can simply say X=RND if you like.   The statement RANDOM
     is also supported, to   reseed the   random number
     generator.

DIM

     You can DIMension up to 20 arrays in a program to be compiled
     with BASIC/S - they can be integer, real or string, as
     distinguished by %, $, etc. The array names may be any
     length (up to 255) with every character significant.
     ONLY letters A-Z should be used for the array names.
     Thus

         DIM ARRAY(20,7),ST$(15),NUM%(50)

     You may have one or two dimensions for each array - no
     more. DO NOT use BASIC keywords in your array names.
     Be careful about your available array space - BASIC/S
     will tell you if your array space will overlay BASIC/S
     data areas or the currently set high memory. It will
     also let you know exactly where your array space lies-
     if the latter number is FFFF, look out! That means that
     your arrays are dimensioned too large (almost certainly).
     If this happens, try recompiling with a start address of 56000;
     this will give you about 19.75 K of space for your arrays,
     as it puts your /CMD file in high memory instead of low.
     Still, 19.75K is only enough room for a string array of
     dimension 79 (79 * 256 = 20,224). With real and integer
     arrays, you can use much larger dimensions.

Syntax for using array elements :
     For the most part, you can use your array variables just like
     any other variables; and you may always use integer

constants (as well as variables) for the subscripts).
Thus

```
        READ NUM%(I%)
        INPUT ARRAY(7)
        PRINT ST$(U%);
        A$=LEFT$(ST$(5),NUM%(I%))
```

The exceptions are as follows :
When an array element is on the left hand side of the
'=' sign, the right hand side MUST be a simple variable
of the same type - no constants or expressions allowed.
Thus ST$(1)="HELLO" is not allowed; you would need to
set H$="HELLO" and then ST$(1)=H$.
Also, any statement that references an array element
should contain NO numeric constants of any kind, except
for (possibly) subscripts to the array itself.
One exception here is that array elements may be
compared via the IF statement, and the line number
reference will not be misconstrued. So

```
        IF ST$(1)<ST$(I%) THEN 75
```

is OK; just be sure to follow the syntax in all other
respects. But something like

```
        LINE INPUT#1,ST$(I%)
```

or   PUT 1,L%(I%)
won't work as the '1' will be misunderstood, and translated
to a temporary integer variable, which won't work.

ASSIGNMENT statement :

Following are the allowed forms of the assignment statement.

REAL :

```
        X=Y         (any var=any other var)
        X=const     (var=constant value)
        X=-Y        (var=-other var)
```

X=real expression

        Here (and in the IF statement for real
expressions) is the only place where BASIC/S can handle
complex expressions. A "real expression" is defined as
any   combination   of   the   real   variables   A-Z,
+,-,*,/, ,(,), and the built-in functions SIN, COS,
TAN, ATN,LOG, EXP, SQR, and ABS, and up to 4 constants.
    Thus :

```
        Y=5*SQR(Z*SIN(2*X+C)), for example.
```

Be careful with constants - you may only have 4
"active" constants at one time (for each var type),
and this includes not only obvious constants, but
also unary minus signs - thus Z=2*(-X) would have
two constants (it would be translated into .2*(0-X)).
Important note -- if you divide by a
product, be careful. BASIC/S will interpret A/B*C as

A/(B*C) rather than the usual (A/B)*C. This is due to
the right to left parsing algorithm that is used.   Use
parantheses if in doubt. Another point is this: If
you calculate X Y (X to the Yth power), this is done a
little differently than in Level II -- it is calculated
as EXP(Y*LOG(X)).   Since LOG(X) is undefined for X<=0,
this will CRASH your BASIC/S /CMD file, whereas BASIC
will   normally handle it if it makes sense as a real
number. So if you want to do such a calculation, you
should check for X being 0 or negative.

X=CSNG(X%)
        Builtin conversion function -- since mixed mode
        arithmetic is not supported, this function is
        needed. May NOT be used in a real expression.
        No constants allowed here.
X%=CINT(X) See above.

INTEGERS :

        Integer arithmetic is limited to +,-,* and only 2
operands allowed on the right hand side. No   builtin
functions for integers. Constants may be used, however.
        Thus

```
        X%=A%*B%
        X%=5-B%
```

Note that unary minus is not allowed here (for variables) ie
X%=-Y%+Z% is no good, while X%=Z%-Y% is OK. Constants may be
negative (as in X%=-5+Y%) and of course you may use unary
minus if the right hand side is a single variable, as in X%=-Y%.

STRINGS :

```
        A$=B$
        A$="constant"
        A$=B$+C$       (simple concatenation)
```

Also we have the builtin
string functions ASC, LEN, CHR$, LEFT$, VAL, RIGHT$,
MID$, STR$, and INSTR. Where numeric arguments are
required in the string functions, simple integer
variables or constants must be used - no expressions.
The actual string arguments cannot be constants, but

```
        A$=LEFT$(X$,2)
```

(for example) would be OK.

Also, expressions must be reduced to their simplest
form -- e.g., concatenation within a function or
function composition is not allowed. Break it down!
        Note: The INSTR function differs from the regular
DISK BASIC one in that no starting position may be
specified -- syntax is just   N%=INSTR(A$,B$).
However, unlike previous versions of BASIC/S, ALL of

B$ is searched for, not just the first character.
     MID$ note -- you can use MID$ on the left hand side
of the = sign, and in that case, you can use either of
the two forms MID$(A$,N%)=B$ or MID$(A$,N%,L%)=B$ --
but they will give the same results, i.e. the length of
B$ is used, L% is ignored in the second form. If the
source string (B$) is null, nothing is done.
     Note III: The INKEY$ function is implemented, and
must be used in the form: A$=INKEY$ (or B$, etc.).
Also, VAL may be used for integers only; i.e.,

                    N%=VAL(A$)

and conversely, STR$ works only on POSITIVE integers
(A$=STR$(N%), where N% is not negative). The
argument to STR$ may be an integer constant as
well.

DISK I/O statements

     Essentially, you have ten disk I/O buffers available
for use (0-9), all of which may be used for sequential access,
and two (1 and 2) of which may be used for random access. Here
are the specifics :

OPEN

     The OPEN statement is essentially that of disk BASIC,
     except that the filespec must be a string variable and
     not an expression in quotes. Syntax is

          OPEN"m",b,F$<,r>
     where m = mode = I,O,R, or E
          b = buffer = (0-9) (constant only)
               (must be 1 or 2 for direct access)
          F$ = filespec (variable only)
          r = logical record length (optional -- may be
               either an integer constant or an integer
               variable).

     BASIC/S makes few restrictions on your use of the
disk   I/O statements, so be careful.   For example, if
you wanted to open a sequential file with an LRECL of
_6, you could. However, you would probably be well
advised to stick to direct access files for this!
     OPEN"E" is like OPEN"O" except you start out
positioned at the end of the file.

Sequential I/O is done with the LINE INPUT# and PRINT#
statements.   Just specify a buffer number adjacent to the #,
and you are ready to go. Only a simple string variable may
     be input or output, although PRINT#1,A$; will disable
     the carriage return.

Random disk I/O is accomplished via the following :

FIELD

     You must field your buffer in order to communicate
     between your strings and the disk file being accessed.
     Syntax is :

               FIELD 1,nn AS A$,mm AS B$, ...

     -- the buffer can be 1 or 2, the strings can be any of
     A$ thru Z$ (no array references allowed here!), and
the numbers 'nn', 'mm' etc must be
     integer constants (1-255 -- 0 is not allowed). Also you
     can't really use multiple FIELD stmts for the same file
     -- the second will override the first. Moreover, the
     statements to process a random access file must be
     statically nested -- i.e. do not GOSUB or GOTO a later
     line to FIELD a buffer and then return to do your LSETs
     and PUTs, etc.   Just OPEN the file, FIELD the buffer,
     process it, and CLOSE it, without GOSUBS and GOTOS.
     (At least, don't branch anywhere outside the range
      of statements between the OPEN and CLOSE stmts).

LSET

     To place your strings into the buffer prior to being
     PUT to the disk, use LSET. Thus

               LSET A$=B$ (spacing critical!)

     where A$ is one of the strings mentioned in your FIELD
     statement. If LEN(B$) is less than that of the field
     variable A$, it will be filled out with spaces in the
     buffer.   If greater, only the leftmost portion of B$
     (for the fielding length of A$) will be in the buffer.

PUT

     Syntax is PUT b,N% where b is the buffer number (1 or
     2) and N% is any integer variable, containing the
     record number to be put. The record number variable is
     not optional.

GET

     As in GET 1,R% -- gets the R%th record from the disk
     file, and places its contents into the string variables
     mentioned in the FIELD statement.

LOF   The LOF function is implemented and syntax is

                        N%=LOF(b)

where b is the buffer number (1 or 2 -- must be a
constant).   This returns the number of records in the
currently open file with buffer b.

CVI and MKI$

        For   convenience   in   reading and   writing   integers
from/to direct   access files, these   functions   are
implemented as in TRSDOS. In case you were mystified
as to exactly what they did -- well, if the integer N%
has the 2 byte representation (L,H), then MKI$(N%) is
just CHR$(L)+CHR$(H).   CVI just does the exact reverse.
As with most BASIC/S functions, these may be used only
with simple integer/string variables.
        Also implemented (completely similarly) are CVS
and MKS$.   Since BASIC/S doesn't support double
precision, CVD and MKD$ are not implemented.

CLOSE

        There is no global close in BASIC/S -- you must mention
the buffer number. Thus,

                        CLOSE 5

would close the file with buffer number 5. If you close
a file that isn't open, you will bomb out with 'FILE
NOT OPEN'.

EOF

        This isn't a function as such; it is to be used in a
special form of the IF statement to check for EOF when
inputting from a sequential file. Simply say

                        IF EOF(b) THEN 200

(or whatever line number) to check for end of file on
buffer b (0-9)

                  BASIC/S Memory Map

Following is a map of memory from 5200H up to HIGH$, showing
how BASIC/S uses the memory in your TRS-80 (48K):

```
/CMD file in low mem                    in high mem

5200 -----------                        --------------



      your /CMD file                    Array space (20K)



A100 -----------                        --------------



      This area is always reserved for BASIC/S variables
        and DCB's.



D7D8 ----------                               ---------------



            Free area for your own use (e.g. USR routines).



DAC0 -----------                              ---------------


                                        /CMD file
      Array space
      (DAC0 to HIGH$)



HIGH$----------                         ---------------

========================================================================
```

        --DISCLAIMER OF WARRANTIES & LIMITATIONS OF LIABILITIES --

We have taken great care in preparing this package. We make no expressed
or implied warranty of any kind with regard to this manual or to
BASIC/S. In NO event shall we be liable for incidental or consequential
damage in connection with or arising out of the performance of this
program.

BASIC/S (c)1981 by Bill Stockwell and Quality Software Distributors

                Questions should be addressed to:

                        Bill Stockwell
                        3700 Wakeforest Dr. # 43
                        Houston TX 77098
                        (713) 520-8695
                        Mnet 70070,320

        Bill Stockwell may also be reached on the QSD Sig
        on MicroNet. Leave a message to 70001,610 for info.

Published by: Quality Software Distributors
                11500 Stemmons Expressway Suite 104
                Dallas, Texas 75229

TRS-80 and TRSDOS are registered copyrights of the TANDY CORP.
MOD I and MOD III are also trademarks of the TANDY CORP.
LDOS is a registered trademark of Logical Systems, Inc.
Newdos and Newdos/80 are trademarks of Apparat
Dosplus is a trademark of Micro Systems Software